A Predictive Method for Providing Fault Tolerance in Multi-Agent Systems

Alessandro de Luna Almeida, Samir Aknine, Jean-Pierre Briot, Jacques Malenfant
Université Pierre et Marie Curie-Paris6,UMR 7606, Paris, F-75005 France
{Alessandro.Luna-Almeida, Samir.Aknine, Jean-Pierre.Briot, Jacques.Malenfant}@lip6.fr

Abstract

The growing importance of multi-agent applications and the need for a higher quality of service in these systems justify the increasing interest in fault-tolerant multi-agent systems. In this article, we propose an original method for providing dependability in multi-agent systems through replication. Our method is different from other works because our research focuses on building an automatic, adaptive and predictive replication policy where critical agents are replicated to avoid failures. This policy is determined by taking into account the criticality of the plans of the agents, which contain the collective and individual behaviors of the agents in the application. The set of replication strategies applied at a given moment to an agent is then fine-tuned gradually by the replication system so as to reflect the dynamicity of the multi-agent system. We report on experiments assessing the efficiency of our approach.

1. Introduction

The notion of agent (and multi-agent systems) is getting increased attention as a very promising approach for designing and building future cooperative distributed applications (e.g., crisis management systems, air traffic control, industrial plant automation, e-commerce, communication network management…). Being distributed systems, MASs are susceptible to the same faults that any distributed system is susceptible to, such as software bugs, system crashes, shortage of resources, slow downs or failures in the communication links [1].

In order to prevent that a system stop working properly due to the occurrence of faults, many fault tolerance approaches have been proposed, some more curative e.g., based on exception handling and cooperative recovery [2], and some more preventive, notably based on the concept of replication, i.e. creation of copies of a component in distant machines.

As discussed by [3], software replication in distributed environments has some advantages over other fault-tolerance solutions: it provides the groundwork for the shortest recovery delays, it is less intrusive with respect to execution time, and it scales much better. As we show in the paper, our solution is furthermore transparent, as the task of deciding what entities to replicate and how to parameterize replication is handled automatically.

In most cases, replication is decided and applied statically, before the application starts. However, recent applications, especially those designed as multi-agent systems, can be very dynamic because of the process of reallocation of tasks, flexible organizations, replanification, changes in the roles of the agents, etc. It is thus very difficult, or even impossible, to identify in advance the most critical software components of the application.

Consequently, it is necessary to replicate in an automatic and dynamic way. This involves the study of mechanisms to determine when to replicate the agents, which agents are to be replicated, the quantity of replicas to be made and where to deploy those replicas.

In this paper, we will introduce our approach to building reliable multi-agent systems. It is based on the concept of criticality, a value (evolving in time) associated to each agent in order to reflect the effects of its failure on the overall system. This value is calculated using the plans of the agent, i.e., the actions that the agent has planned to execute in the near future.

A plan-based fault-tolerant mechanism acts as a promising preventive method since it takes into account the prediction of the future behavior of the agents and their influence over the other agents of the society.

The remainder of this paper is organized as follows. Section 2 defines the fault tolerance problem we deal with in this paper. Section 3 explains how the plans of the agents can be used as an approach to this problem. Section 4 describes the general architecture of the experimental platform. Section 5 shows some preliminary results. Section 6 provides an overview of the state of the art. Finally, in section 7 we present our conclusions and perspectives for future work.

2. Problem Definition

The fault tolerance problem described in this paper considers a set of agents $S = \{Agent_1, Agent_2, ..., Agent_n\}$ that have to complete a set of tasks. For example, consider a set of agents called *assistants*, which elaborate plans for a patrolling task (see Fig. 1). The patrol is, in this case, performed by a set of autonomous agents, called *patrollers*. Each patrolling agent will interact with its

Proceedings of the IEEE/WIC/ACM International
Conference on Intelligent Agent Technology (IAT'06)
0-7695-2748-5/06 $20.00 © 2006 IEEE

corresponding assistant agent in order to discover the sequence of sites which must be visited. A predetermined priority is associated to each site.

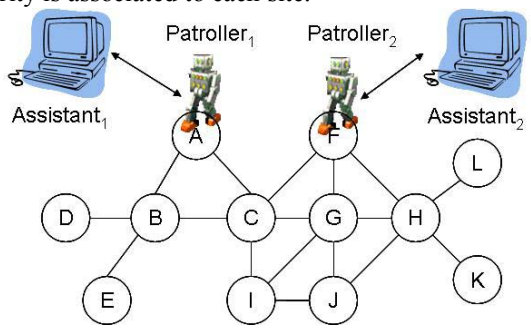

Figure 1. Example of the patrolling task

While trying to accomplish their tasks, agents can stop executing. In this work, we consider the crash type of failures, that is when a component stops producing output. It is the simplest type of failure to contend with. However, in various cases our solution allows to deal with other types of failures (omission, timing, byzantine). They are currently being investigated, but will not be considered in this paper.

To minimize the impact of failures, agents can be replicated. A replicated software component has representations (replicas) on two or more hosts [3]. The two main types of replication protocols are: active replication, in which all replicas process concurrently all input messages ; passive replication, in which only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency.

Replicating every agent is not a feasible approach since not only the available resources are often limited, but also the overhead imposed by the replication could degrade performance significantly. The problem consists in finding a replication scheme which minimizes the probability of failure of the most critical agents. This scheme must also be revised over time, considering that the multi-agent execution context of tasks is dynamic and, thus, the criticalities of the agents vary at runtime.

3. Our Plan-Based Criticality Assessment Method

In our approach to solve the problem defined in the last section, we consider that each agent of the system knows which sequence of actions (plan) must be executed in order to accomplish its current task. Since unexpected events may occur in dynamic environments, agents usually interleave planning and execution. Consequently, their plans are established just for the short term. We assume that at each given instant of time the agent is executing at most one action.

Using the same approach established by [4], we represent the plan of an agent as a directed acyclic AND/OR graph where each node represents an action. The nodes are connected by AND or OR edges. A node n which is connected to k nodes (n_1, n_2, ..., n_k) by means of AND edges represents an action A_n after which *all* the actions A_{n1}, A_{n2}, …, A_{nk} will be executed. However, if a node n is connected to k nodes (n_1, n_2, ..., n_k) by means of OR edges, it suffices that *at least one* of the actions A_{n1}, A_{n2}, …, A_{nk} be executed after the execution of the action A_n.

In the example of Fig. 2, we show two patrolling plans elaborated by the assistant agents. For legibility purposes, let us denote the action "*visit the site X*" simply as "*X*". After performing the action *A*, $Agent_1$ needs to have both *B* and *C* executed in order to accomplish its plan. However, after *C*, only one of *D* or *E* needs to be performed so that $Agent_1$ accomplishes its plan.

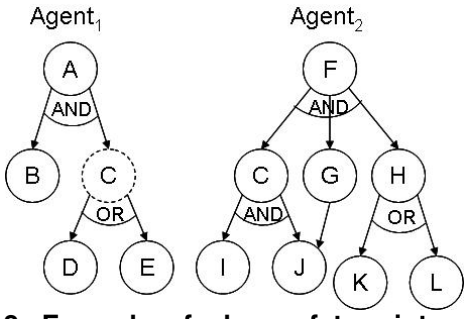

Figure 2. Example of plans of two interacting agents (dashed circles represent external actions)

Definition 1: An *external* action is an action belonging to the plan of an agent which will be executed by others. For example, consider the action *C* belonging to the plan of $Agent_1$ in Fig. 2. Since this action is performed by $Agent_2$, it is an external action in the current plan of $Agent_1$.

Definition 2: A *terminal* action is an action after which no other known action will be performed. In Fig. 2, *B*, *D*, *E*, *I*, *J*, *K*, and *L* are terminal actions.

3.1. Agent Criticality

The criticality of an agent at any time can be calculated based on the criticalities of the forthcoming actions which belong to its plan. An agent which executes important actions must be considered critical. In a given time *t*, the criticality of the agent will be given by the relative criticality of the current root of its plans' graph.

Before defining the relative criticality of an action, let's first introduce the concept of absolute criticality. The *absolute criticality (AC)* of an action is defined without taking into account the current plans of the agents. It is given a priori by the system designer and can be determined in function of a number of factors:

- Number of agents capable of performing the action: an action that can be done by many agents can be

Proceedings of the IEEE/WIC/ACM International
Conference on Intelligent Agent Technology (IAT'06)
0-7695-2748-5/06 $20.00 © 2006 IEEE

considered not too critical, since it is probably easier to reschedule it, if it ever fails, than if only a few agents were capable of performing it.

- Resources required for the execution of the action: the size of the set of required resources can also be used to determine the absolute criticality. For example, under some circumstances, actions which are expected to take too long could be considered more critical than short ones.

- Semantic information: the system designer can use semantic information to determine the criticality of the action, since, depending on the field of application, some actions are more important than others. For example, the absolute criticality of the action *"visit the site s"* in the case of the patrolling task is given by the priority of the site *s*.

The *relative criticality (RC)* of an action executed by an agent (possibly jointly with other agents) estimates the impact of its failure to the multi-agent system as a whole. The RC depends on the absolute criticality of the action and on the usefulness of its results to all the agents which depend on it to perform their tasks.

The relative criticality is calculated as follows:

- For an external action, it is equal to the *local relative criticality (LRC)*. The LRC is obtained using the AND-aggregation function if the action is connected to its children by means of AND edges or the OR-aggregation function if it is connected by OR edges. The parameters of these two functions are the relative criticalities of the children of the action. We use as an AND-aggregation function the sum of its parameters and as OR-aggregation function, the mean of its parameters. If the action has only one child, its LRC is equal to the relative criticality of its child. If the action is terminal (i.e. it has no child), its local relative criticality is equal to zero.

- For a non-external action *a*, its relative criticality is equal to its absolute criticality plus the sum of the local relative criticalities of *a* in each plan to which it belongs.

Table 1 shows the relative criticalities of each action in the example of Fig. 2 if the corresponding absolute criticalities are considered.

Table 1. Calculation of Criticality

Action	Absolute Criticality	Relative Criticality
A	4	15
B	8	8
C (Agent1)	4	3
C (Agent2)	4	13
D	5	5
E	1	1
F	6	30
G	3	7
H	2	4
I	2	2
J	4	4
K	3	3
L	1	1

In order to obtain those values for the relative criticalities, the method previously described is used. For example, the action *B* belonging to the plan of $Agent_1$ is a non-external action. Then its relative criticality is calculated by adding its absolute criticality with its local relative criticality. Since it is a terminal action, its local relative criticality is equal to zero.

$$RC(B) = AC(B) + LRC(B, Agent_1) = 8 + 0 = 8 \qquad (1)$$

However, we calculate the relative criticality of the action *C* in the plan of $Agent_1$ differently because it is an external action (it will be executed by $Agent_2$). In this case, the relative criticality is simply equal to the value of the local relative criticality. In order to calculate the LRC of *C* in $Agent_1$'s plan, we use the mean aggregation function with the relative criticalities of the children of action *C* (namely *D* and *E*) as parameters.

$$RC(C\_Agent_1) = LRC(C, Agent_1) = Mean(RC(D), RC(E)) = Mean(5, 1) = 3 \qquad (2)$$

In dynamic and unreliable environments, actions with a late start time will be executed less possibly than actions with an early starting time, since the plans can change or failures can happen. Consequently, we have also refined this strategy by considering the expected starting time of actions.

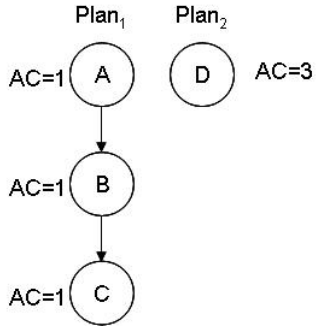

Figure 3. Impact of time in the criticality of actions

In Fig. 3, the Action *A* of $Plan_1$ and the action *D* of $Plan_2$ will have a relative criticality of *3*. However, since it is not certain that actions *B* and *C* will be executed, the action *D* should be more critical than the action *A*. Hence, we propose another approach to calculate the relative criticalities, where we multiply the relative criticality of the actions by a factor which decreases along time, taking into account the expected time that the action will start to be executed.

Let *t* be the estimated starting time of the action and RC_{old} its relative criticality calculated using the previous approach. Then we calculate the relative criticalities in the new approach (RC_{new}) using the following exponentially decreasing function:

$$RC_{new} = RC_{old} / b^t, \text{ where } b \geq 1 \qquad (3)$$

We intend to study other functions besides the exponential one, such as a linear or hyperbolic decay.

We compute the estimated starting time of the actions using a topological sorting in the graph (top-down) considering the elapsed times of the antecedents and siblings' actions [5].

In the example of Fig. 3, if we consider that the duration of all the actions is equal to three units of time, and the amortizing base b is equal to e, the following relative criticalities would be obtained:

Table 2. Calculation of Criticality considering time

Action	Relative Criticality
A	$RC(A) = (AC(A) + LRC(A, Agent))/e^{ta} = (AC(A) + RC(B))/e^{ta} = (1 + e^{-3} + e^{-9})/e^0 = 1 + e^{-3} + e^{-9}$
B	$RC(B) = (AC(B) + LRC(B, Agent))/e^{tb} = (AC(B) + RC(C))/e^{tb} = (1+ e^{-6})/e^3 = e^{-3} + e^{-9}$
C	$RC(C) = AC(C)/e^{tc} = 1/e^6 = e^{-6}$
D	$RC(D) = AC(D)/e^{td} = 3/e^0 = 3$

Using this new mechanism for calculating the relative criticalities, the action A in $Plan_1$ is less critical than the action D in $Plan_2$, as desired.

3.2. Revision of the Criticality

Since multi-agent systems are often dynamic and non-deterministic, it is not possible to know in advance the complete plan of the agent. Actually, during the execution of multi-agent plans, one or more agents might determine that the context has changed so much that the agents' partial plan should be modified.

Consequently, the initial criticality of the agents in the instant $t = 0$ is quite precise, but it needs to be updated along time. The question is when and how to update those criticalities. We propose two main types of strategies to revise the criticality: *time-driven strategies* and *event-driven strategies*.

Time-driven strategies are based on local clocks associated to each agent. Whenever the clock alarms, the criticality of the corresponding agent is re-evaluated. The interval of time between two consecutive alarms can be *fixed* or *variable*. Using an initial approach, at each fixed interval Δt, the clock will sound the alarm and the criticality will be updated. The value of Δt could be variable so as to reflect the dynamicity of the system. If this is the case, the length of the interval is initially set to a predefined value. It is reduced if a substantial modification in the criticality has been noticed in the last interval of time or, inversely, it would be increased if almost no change has been observed in the criticality.

Event-driven strategies are based on critical events that might change the criticality in a substantial way. Whenever one of these events is detected, the criticality is updated. There are two main types of events: those which depend on the application (completion of an action, changes in the plan of the agent, …) and those related to failures (failure of an agent or a machine).

3.3. Agent Replication Mechanism

Once estimated the criticality of the agents (using the strategies described in the last section, for example), one may ask which agents should be replicated and where to deploy the replicas. In order to determine which agents to replicate, Guessoum et al [6] proposed an agent replication mechanism which, at each interval of time Δt, recalculates and updates the number of replicas that each agent must possess. It is directly proportional to the criticality of the agent and the number of replicas available and inversely proportional to the sum of criticality of all agents in the system.

One problem with this technique of calculating the number of replicas that should be given to each agent is that it does not address the problem of where deploying efficiently the replicas. In other words, it does not take into consideration the future failure probability of the replica. In fact, it is better to have only one replica which will have in the future an almost zero probability of failure than having many replicas which are not reliable.

Hence, we will propose another mechanism of replica allocation, which considers the probability of failures. In this new mechanism, we define the *value* of the replica r_k (denoted by v_k), as the probability that it will not crash. A value of one will be attributed to a completely reliable resource, whereas an unreliable one shall have a near zero value.

The probability of failure of a given set of replicas $R = \{r_1, r_2, \ldots, r_n\}$, is given by:

$$P(Failure(R) = 1) = (1-v_1) \times (1-v_2) \times \ldots \times (1-v_n) \qquad (4)$$

Let S be the sum of the values of all the replicas in the system. Then, an agent $Agent_i$ is allowed to be replicated using a total value of replicas (t_i) proportional to the percentage of its criticality (c_i) with respect to the sum of agents' criticalities (C), as given by the equation:

$$t_i = c_i \times V / C \qquad (5)$$

The system of replication will then allocate to the agent the set of replicas $R = \{r_1, r_2, \ldots, r_n\}$, such that $v_1 + v_2 + \ldots + v_n \leq t_i$ and its probability of failure is minimal among all the possible sets of replicas.

One can apply the same possible strategies used as the agent criticality update policy (time-driven or event-driven) to decide when to re-calculate the values of t_i. For instance, one can use a variable window of time Δt for

each agent $Agent_i$. If the quantity of replicas (whose total value does not exceed t_i) that the agent $Agent_i$ can acquire does not change significantly, the window of time Δt can be increased, otherwise it is decremented. Another possibility is to recalculate the value of t_i whenever the value of c_i is updated.

4. Architecture and Implementation

To implement the agent replication mechanism described in the previous section, we used the framework DARX (Dynamic Agent Replication eXtension) [7].

DARX relies on the notion of replication group (RG). Every agent of the application is associated to an RG, which DARX handles in a way that renders replication transparent to the application at runtime. Each RG has exactly one ruler, which communicates with the other agents. Other RG members, referred to as subjects, are kept consistent with their ruler according to the replication strategies. Several different strategies, ranging from passive to active, may be applied within a replication group. The number of subjects and the replication strategy may be adapted dynamically.

DARX provides atomic and ordered multi-cast for the replication groups' internal communication. Messages between agents are also logged by each replica, and sequences of messages can be re-emitted for recovery purposes.

As shown in Fig. 4, DARX offers several services. Failure detection enables to suspect host and process failures based on a hierarchy of adaptive failure detectors. Naming and localisation provides a means to supply agents and their replicas with unique identifiers throughout the system, and to retrieve their location whenever the application requires it.

DARX is coded in Java 1.4 and uses RMI as a means to simplify the coding of network issues. It can be easily integrated to any agent platform by means of an interfacing component. Current implementation provides the integration to DIMA and Madkit multi-agent platforms.

We implemented the proposed replication mechanism in an adaptive replication control module, which we have coupled to the DARX platform. This module is completely distributed and uses the replication service of DARX to provide a suitable replication scheme for every agent.

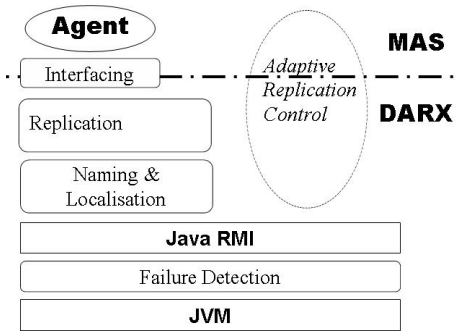

Figure 4. DARX framework design

5. Experimental Results

We are currently conducting experiments, whose preliminary results we summarize in this paper. In our experiments, each agent has to accomplish its own sequence of 5 plans, one at a time, each with 10 actions. The average duration of actions is of 2 seconds. We repeated ten times each experiment (the results shown are the mean of those several runs). We maintained the same sequence of plans and actions that each agent must execute in those runs.

In the first place, we ran each experiment considering a completely reliable environment (no failures) and calculated the CPU time (in milliseconds) required for the completion of all the plans by all the agents. Fig. 5 shows the effect of changing the number of agents on the CPU time required (y-axis) by our replication mechanism and by the execution of the multi-agent system with no replication at all. Whenever replication is present, the number of replicas available at the machine is half of the number of agents. One can notice that using no replication always outperform our replication mechanism, but the overhead of our mechanism is negligible (less than 4%).

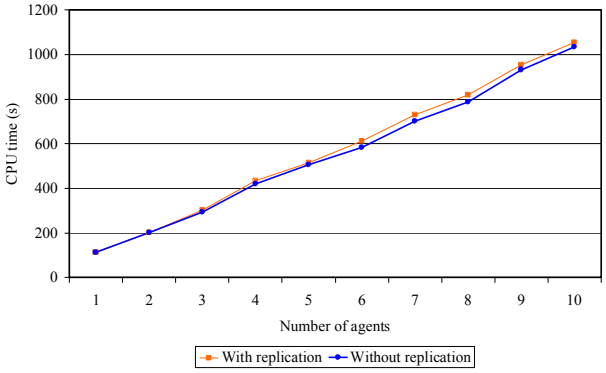

Figure 5. Impact of the usage of replication on CPU time

In order to assess the quality of a replication mechanism, we considered the sum of the absolute criticalities of the actions which were executed with success using the corresponding mechanism. During the

Proceedings of the IEEE/WIC/ACM International
Conference on Intelligent Agent Technology (IAT'06)
0-7695-2748-5/06 $20.00 © 2006 IEEE

execution of each experiment, at each interval of 2s and for each agent, a failure generator will cause the agent to fail with a probability equal to its probability of failure given by Eq. 4. Whenever an agent fails (because all its replicas failed), its current plan fails, the agent is restarted with its next plan and all the replicas which were allocated to it are made available for use.

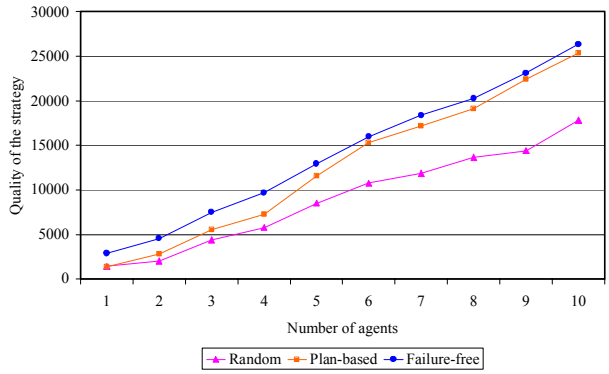

Figure 6. Quality of the replication mechanism used vs. a failure-free execution

Fig. 6 shows the maximum quality that could be obtained (in a completely reliable environment) compared to the quality of our strategy of replication and to a random one, which allocates randomly each replica available. We varied the probability of failure of the replicas, but due to space constraints, only a fixed value of 0.3 is reported in Fig. 6. The results are encouraging in the sense that our strategy is more accurate to determine and replicate the most critical agents than the random one. In fact, the probability that a critical agent fails with our strategy is lower than with a random strategy. Additionally, the quality of our mechanism is quite close (80% at average) to the maximum value that could be obtained in a failure-free execution.

The results show here are just partial. In fact, we envisage following the work of experimentation and evaluation by integrating other methods of the domain with which we intend to compare our approach.

6. Related Work

Several approaches have addressed the problem of fault tolerance. In fact, many toolkits include replication facilities to build reliable applications. However, most of them are not quite suitable for implementing large-scale, adaptive replication mechanisms.

The first type of solution has been introduced by using reactive MASs, such as those based on the metaphor of ant nests [8]. An ant nest is composed of a set of simple agents which act in a complex environment and which can exhibit intelligent and adaptive behaviours. They are robust and fault tolerant systems a priori, since they are founded on the redundancy of similar reactive agents. The fault of an agent does not affect, thus, the global system functioning. However, this redundancy is not a priori guaranteed to cognitive MASs, since cognitive agents can present very different, dependant and complex behaviours.

Hägg [9] proposes an approach to the problem in which sentinel agents monitor inter-agent communication, build models of other agents and take corrective actions. Since the sentinels analyze the entire communication going on in the system to detect state inconsistencies, it would be far too expensive in terms of computation and communication to take total control of possible fault situations and global consistency. Additionally, sentinels are themselves points of failures.

Decker et al [10] and Shen et al [11] offer dynamic cloning of specific agents in multi-agent systems. But their motivation is different, in view of the fact that their objective is to improve the availability of an agent if it is too congested (load balancing). Fault tolerance aspects are not addressed.

Fedoruk and Deters [1] also use replication to improve fault tolerance. Their work implements the passive strategy (hot-standby) of replication in a transparent way using proxies. All messages going to and from a replicate group are funnelled through the replicate group message proxy. Kraus et al [12] define the problem of fault tolerance as a deployment problem and propose a probabilistic approach to deploy the agents in a multi-agent application. The main problem of these two works is that replication is applied statically before the application starts. This is not desirable in the case of dynamic multi-agent applications because the criticality of agents may evolve dynamically during the course of computation.

In distributed computing, many toolkits include replication facilities to build reliable application. However, many of products are not enough flexible to implement an adaptive replication. MetaXa [13] implements in Java active and passive replication in a flexible and transparent way. Authors extended Java with a reactive meta-level architecture. However, MetaXa relies on a modified Java interpreter. GARF [14] uses a reflexive architecture to allow the development of fault-tolerant Smalltalk applications. It provides different replication strategies, but, it does not offer adaptive mechanisms to apply these strategies.

There are other software infrastructures for adaptive fault tolerance [15]–[17] where existing strategies can be dynamically changed. Nevertheless, such a change must have been devised by the application developer before runtime or the modifications must be specified and applied in a non-automatic way during the execution of the system.

As we said before, Guessoum et al [6] propose an adaptive replication mechanism based on the criticality of the agents. Their work uses system-level information

Proceedings of the IEEE/WIC/ACM International
Conference on Intelligent Agent Technology (IAT'06)
0-7695-2748-5/06 $20.00 © 2006 IEEE

(communication load and processing time) as well as semantic-level information (the role taken by an agent in an organization, e.g., role of broker, manager) to calculate the criticality of an agent. However, they do not consider the probability of failures of replicas and do not address the problem of where deploying the agents. Additionally, the future behavior of the agents is not taken into account when calculating the criticalities.

7. Conclusion

Large-scale multi-agent systems are often distributed and must run without any interruption. To make these systems reliable, we proposed an original predictive method to evaluate dynamically the criticality of agents. Our approach takes profit of the specificities of multi-agent applications and analyses the agents' plans to determine their importance to the system. This approach allows us to obtain a more precise value of the criticality since it takes into account the future behavior of the agents. The agent criticality is then used to replicate agents in order to maximize their reliability and availability based on available resources.

The proposed mechanism was implemented over the DARX replication platform. We have tested it and we believe that our current results are promising. In fact, the algorithms have a negligible overhead and provide a satisfactory reliability.

One of the perspectives of this work is to better formalize the problem of fault tolerance in multi-agent systems and its evaluation measures, in order to compare the different proposed strategies with an optimal one.

Additionally, different real-world applications will be implemented and used to validate our approach. Two applications are for the moment envisaged: the personal meeting assistants and the patrolling agents [18].

References

[1] A. Fedoruk, R. Deters, "Improving fault-tolerance by replicating agents", In *Proc. AAMAS-02*, Bologna, Italy, 2002, pp. 737-744.

[2] A. Romanovsky, C. Dony, J.L. Knudsen, and A. Tripathi (eds). "Advances in Exception Handling Techniques", LNCS 2022, Springer, 2001.

[3] R. Guerraoui and A. Schiper, "Software-based Replication for Fault Tolerance", *IEEE Computer*, vol. 30, no. 4, 1997, pp. 68-74.

[4] B. Horling et al., "The TAEMS White Paper", January 1999.

[5] Hillier and Lieberman, "Introduction to Operations Research". Third Edition. Holden-Day Inc, pp. 246-259.

[6] Z. Guessoum, J.-P. Briot, O. Marin, A. Hamel, P. Sens, "Dynamic and adaptive replication for large-scale reliable multi-agent systems", In *Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, LNCS 2603, April 2003, pp. 182-198.

[7] O. Marin, P. Sens, J.-P. Briot, Z. Guessoum, "Towards adaptive fault-tolerance for distributed multi-agent systems", In *Proc. of ERSADS'2001*, Bertinoro, Italy, 2001, pp. 195-201.

[8] A. Drogoul, J. Ferber, "Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies", In *Decentralized A.I.*, vol. 4, Elsevier North-Holland, 1992.

[9] S. Hägg, "A sentinel approach to fault handling in multi-agent systems", In *Proc. of the Second Australian Workshop on Distributed AI*, Cairns, Australia, August 27, 1996.

[10] K. S. Decker, K. Sycara, "Intelligent adaptive information agents", *Journal of Intelligent Information Systems*, vol. 9, 1997, pp. 239 - 260.

[11] W. Shen, D. H. Norrie, "A Hybrid Agent-Oriented Infrastructure for Modeling Manufacturing Enterprises". In *Proceedings of Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Canada, 1998, pp. 1-19.

[12] S. Kraus, V.S. Subrahmanian, N. Cihan, "Probabilistically survivable MASs", In *Proc. of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 789-795.

[13] M. Golm, "MetaXa and the Future of Reflection", In *OOPSLA Workshop on Reflective Programming in C++ and Java*, Springer Verlag, 1998, pp. 238–256.

[14] R. Guerraoui, B. Garbinato, K. Mazouni, "Lessons from Designing and Implementing GARF". In *Proc. Object-Based Parallel and Distributed Computation*, LNCS 1101, 1995, pp. 238-256.

[15] Z. Kalbarczyk, S. Bagchi, K. Whisnant, R.K. Iyer, "Chameleon: a software infrastructure for adaptive fault tolerance", *IEEE Transactions on Parallel and Distributed Systems*, 1999, pp. 560-579.

[16] M. Cuckuern et al, "AQuA: an adaptive architecture that provides dependable distributed objects", In *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98)*, West Lafayette, Indiana, October 20-23, 1998, pp. 245-253.

[17] F. Favarim, F. Siqueira, J. S. Fraga, "Adaptive fault-tolerant CORBA components", In *Middleware Workshops 2003*, pp. 144-148.

[18] A. L. Almeida et al, "Recent advances on multi-agent patrolling," In *Proc. SBIA 2004*, pp. 474-483.

Proceedings of the IEEE/WIC/ACM International
Conference on Intelligent Agent Technology (IAT'06)
0-7695-2748-5/06 $20.00 © 2006 IEEE